U.S. Army
Research Institute of
Environmental Medicine
Natick, Massachusetts

TECHNICAL REPORT NO. T17-09
DATE January 2017

DETERMINING THE CENTER PATH OF GROUND SURFACE LIDAR DATA

United States Army
Medical Research & Materiel Command

DISCLAIMER

The opinions or assertions contained herein are the private views of the authors and are not to be construed as official or as reflecting the views of the Army or the Department of Defense. The investigators have adhered to the policies for protection of human subjects as prescribed in Army Regulation 70-25 and SECNAVINST 3900.39D, and the research was conducted in adherence with the provisions of 32 CFR Part 219. Citations of commercial organizations and trade names in this report do not constitute an official Department of the Army endorsement or approval of the products or services of these organizations.

USARIEM TECHNICAL REPORT TR17-09

DETERMINING THE CENTER PATH OF GROUND SURFACE LIDAR DATA

Gary P. Zientara, Ph.D.
Maxwell Rome, B.A.
Stephen P. Mullen, M.S.
William R. Santee, Ph.D.

Biophysics and Biomedical Modeling Division

January 2017

U.S. Army Research Institute of Environmental Medicine
Natick, MA 01760-5007

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-01-2017	Technical Report	2016 - 2017

4. TITLE AND SUBTITLE

THERMAL-WORK STRAIN AND ENERGY EXPENDITURE DURING MARINE RIFLE SQUAD OPERATIONS IN AFGHANISTAN (AUGUST 2013).

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Gary P. Zientara, Ph.D.
Maxwell Rome, B.A.
Stephen P. Mullen, M.S.
William R. Santee, Ph.D.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Biophysics and Biomedical Modeling Division
U.S. Army Research Institute of Environmental Medicine
Building 42 - Kansas Street
Natick, MA 01760

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Medical Research and Materiel Command,
Fort Detrick, MD 21702

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report describes a computational methodology for determining the central path through ground surface LIDAR 'point cloud' data, i.e. many points located in 3-D space. This approach answers the need for a singular point-to-point path useful for follow-up applications after acquisition of a large magnitude of data by LIDAR ground surveying. The methods employed include a preliminary approximate ordering of the LIDAR coordinates and color data, followed by the computation of minimum enclosing rectangles of subsets of the preordered original data. The centroids of the minimum enclosing rectangles that cover the LIDAR data are identified as final path points, since they are at the center of local LIDAR data and unaffected by the variable LIDAR data density. To provide a more precise description for later applications, the final path is defined to also include points uniformly interpolated between the rectangle centroid coordinates. This approach was applied to USGS-scanned LIDAR coordinates and color data acquired from the Natick Soldier Systems Center (NSSC) Fitness Trail. The path determined from the results of this study will contribute directly to more accurate interpretations of warfighter physiology and performance experiments carried out using the trail

15. SUBJECT TERMS

LIDAR; computational methods; land survey; wayfinding

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Adam W. Potter
Unclassified	Unclassified	Unclassified	Unclassified	41	19b. TELEPHONE NUMBER *(Include area code)* 508-233-4735

Reset

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEGDEMENTS

The authors would like to thank Dr. R. W. Hoyt and Adam W. Potter M.S., M.B.A. for project support, critical review and administrative assistance.

EXECUTIVE SUMMARY

This report describes a computational methodology for determining the central path through ground surface LIDAR ‘point cloud’ data, i.e. many points located in 3-D space. This approach answers the need for a singular point-to-point path useful for follow-up applications after acquisition of a large magnitude of data by LIDAR ground surveying. The methods employed include a preliminary approximate ordering of the LIDAR coordinates and color data, followed by the computation of minimum enclosing rectangles of subsets of the preordered original data. The centroids of the minimum enclosing rectangles that cover the LIDAR data are identified as final path points, since they are at the center of local LIDAR data and unaffected by the variable LIDAR data density. To provide a more precise description for later applications, the final path is defined to also include points uniformly interpolated between the rectangle centroid coordinates. This approach was applied to USGS-scanned LIDAR coordinates and color data acquired from the Natick Soldier Systems Center (NSSC) Fitness Trail. The path determined from the results of this study will contribute directly to more accurate interpretations of warfighter physiology and performance experiments carried out using the trail despite recent renovations.

INTRODUCTION

The main objective of this study was to determine the singular center path given a large point cloud of previously acquired LIDAR-derived ground scan data acquired from a wide scan of a long walking trail, with hundreds to thousands of survey data per square meter. The output of the method is a uniformly ordered geo-referenced data set representing the center line of the trail. These geo-referenced data will be used as part of the contextual information during analysis of human physiologic data gathered as test volunteers traversed the trail.

Light Detection and Ranging (LIDAR) surveying equipment includes a high precision laser scanner and a special Global Positioning System (GPS) system. The LIDAR scanner transmits optical laser light (YAG laser; near infrared; 1064nm wavelength) in pulses and detects their reflection. From the time interval between transmission and return of the reflected beam, the distance to the ground can be computed. That information, together with the GPS information, gives highly accurate GPS ground coordinates of the scanned points.

In early 2013, USARIEM's Biophysics and Biomedical Modeling Division, BBMD, commissioned the US Army Corps of Engineers Engineering Research and Development Center, ERDC, Geospatial Research and Engineering Division to perform a very high resolution geodetic and terrestrial LIDAR survey and scan of the NSRDEC Fitness Trail. This trail is approximately 1.6mi long by 2m wide, and generally follows the perimeter of the Soldier Systems Center in Natick, MA.

The scan was performed by members of the ERDC Topographic Engineering Center's Lidar Support for Dismounted Operations (LASSO) team. A total of 53 files from 59 individual LIDAR scans were produced. These data have a nominal resolution of 4 cm at a 50 meter range and were acquired from a Leica C-10 LIDAR scanner. They were georeferenced using survey-

grade differential GPS instruments. The survey data accuracy as measured by average sums of errors (i.e positional misclosures) is better than 2 cm.

The three-person ERDC team made their base measurements during a three-day period in May 2013, traversing the trail with both LIDAR and GPS instruments using standard land surveying techniques, maintaining line of site from one measurement/scan position to the next.

The raw data were initially processed by first registering multiple scans (trail sections) into one registered model. Using Terascan software the ERDC team then edited out all non-trail points to provide a gridded ribbon representing the trail. The final data set that was acquired included approximately 77,000,000 3D GPS map coordinates, accompanying r,g,b color intensity (i.e. red (r), green (g), blue (b) color intensity values in the 8-bit/datum range 0-255). x and y coordinates are Universal transverse Mercator (UTM) Zone19N easting and northing, respectively. z coordinates are NAVD88 Orthometric heights ("mean sea level").

The points were tiled into 4 quadrants: NW, NE, SW, and SE in order to make the file sizes manageable. Each tile is 250n (E-W) by 400m (N-S) and includes a 10m overlap with contiguous tiles.

Data were delivered in a number of file formats. Because MATLAB (MathWorks Inc.; Natick MA) will be used for subsequent analysis involving these trail data, the non-proprietary, plain text xyzrgb format was chosen as the one best suited to our long term needs. The large volume of data required a method with efficient programming and simple data structures. In determining the singular center path, all interim and final data were processed in a clockwise fashion along the georeferenced data set.

The expected product of the study was to be an ordered set of x,y,z and r,g,b coordinates and values that represented a single center path through the point cloud data.

Notably, the solution to the problem of this study, the identification of a central path through a point cloud of non-uniform density does not fall within the realm of the historically important shortest path solution literature (e.g. [Dijkstra 1959] for undirected graphs, and [Bellman 1958][Ford 1956] for directed graphs with non-negative weights) or graph traversal literature. Also, an adjacency analysis or similar scheme is impractical because they are precluded by their storage requirements.

More modern wayfinding methods have taken into account ever increasing complexity in route finding and planning motivated by strong commercial interest in applications providing traffic mapping and directions. Modern route-finding and directions-giving methods have become increasingly sophisticated, and can account for congestion factors, personal route preferences, the mode of transport (walking/hiking, driving, etc.), navigation using landmarks, and human factors in describing and remembering directions. These benefit from known possible routes that can be represented mathematically by graph structures, thereby able to seek solution in the large computer science repertoire of graph traversal methods. Unfortunately, these too do not apply here.

METHODS

APPROXIMATE ORDERING ALONG PATH

The raw scanned data in this study is only partially ordered, reflecting the circumnavigation of the wide path followed by technicians as the LIDAR data was acquired. This, however, did not represent a global ordering in terms of any one coordinate nor the curve of the path. In addition, the data appeared mixed with data from distant locations along the wide path. Therefore, the data is initially subjected to a random permutation prior to the start of processing, in order that no bias in the initial data ordering entered the ensuing computations.

The first step is to create an approximate ordering of the randomly ordered raw data along the path, grouping coordinates into a partially ordered list. An arbitrary starting point on the path is first chosen. The data are grouped along the path within a circle of radius 2-10m from the starting path coordinate. The center of a new circle is chosen from the remaining points to be considered, a minimum distance (at approximately the distance between coordinates) immediately outside the previous circle, and the grouping process is repeated until the data has been completely.

The final output coordinate list is grouped in sets of coordinates approximately ordered so as to lead from the starting trail coordinate to its end, clockwise. The effect is to traverse the trail in the sector by large sets of points. At this juncture, the points within each circle are not ordered, however, the *point sets* contained in the circles are ordered. This first processing step prepares the partly ordered coordinate list as input

for the next step, which will analyze the point cloud data and use a computational geometry method to compute the path center coordinates.

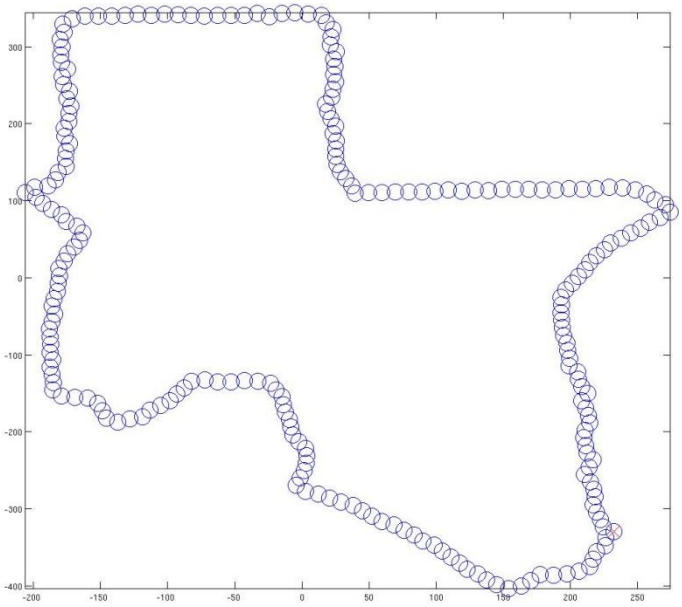

FIGURE 1. Example of circular regions chosen to obtain reordered LIDAR ground scanned point cloud data acquired from the NSSC Fitness Trail. The y axis points North, x axis points East.

Some programming efficiencies in the initial pre-ordering step can be sought if the LIDAR data is represented in cylindrical coordinates (r,θ, z), but Cartesian domain computations proved practical for our example problem.

DETERMINING THE CENTER OF THE PATH

The center of the wide scanned point cloud path is then identified with the path (i.e. a discrete curve formed with many nodes and segments) formed by the centroids of the minimum enclosing rectangles (see, e.g. [Preparata & Shamos 1985]) of the ordered sets of data produced by the first processing step. The centroids of the

minimum enclosing rectangles that cover the LIDAR data are identified as a partial set of the final path points, since they represent the center of local LIDAR data boundaries, and are unaffected by the variable LIDAR data density. Different sets of data, taken from the preordered list, are analyzed in this step, not specifically the sets enclosed by a circle used above in the first step. The number of coordinates in the sets to be considered in this step may differ from group to group if one has split the original data into several smaller groups order to accomplish the processing.

The minimum enclosing rectangle, an object from computational geometry, is the rectangle of minimum area that can be drawn to enclose a set of points in the plane. A number of algorithms exist to compute the minimum enclosing rectangle. MATLAB code of [D'Errico 2012] is used here, embodying the rotating calipers algorithm of [Toussaint 1983].

The aim in this step is to select a set of coordinate points from the partly ordered coordinate list determined in the first step. The selected set of points must be large enough to populate one or two square meters along the scanned path. For each set of coordinates considered, the minimum enclosing rectangle is determined and the x,y coordinates of the rectangle vertices, the area and perimeter are recorded. If for some unknown reason the rectangle perimeter limit exceeds 100m during this computation, the number of coordinates in the coordinate set considered is halved, and the computation is repeated.

The plan is to circumnavigate the overall path via computing large rectangles whose footprint covers the path, and whose sides collectively approximate the side

boundaries of the point cloud path. Given these local properties, we claim the centroid of each computed rectangle represents a point along the singular center path of the (relatively wide) point cloud path. Since the vertices and centroid of the minimum rectangle likely do not fall exactly on LIDAR scanned points, the z coordinate of the centroid is derived from the nearest scanned location.

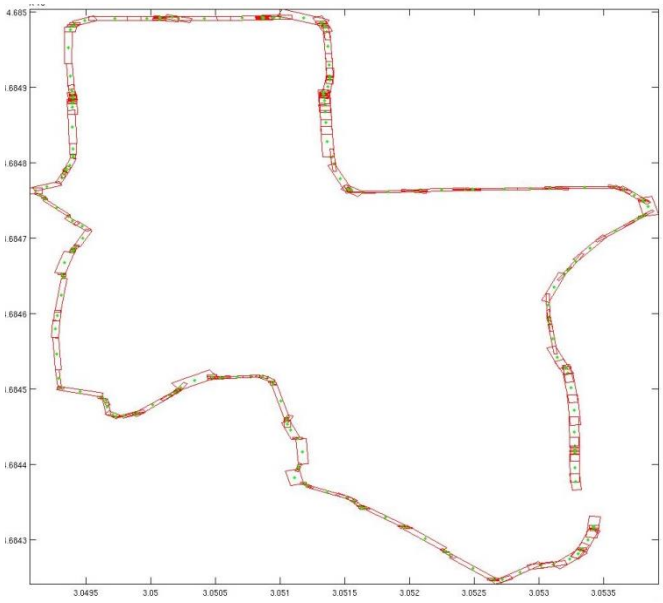

FIGURE 2. Minimum enclosing rectangles (red) and centroids (green) computed from reordered LIDAR ground scanned point cloud data acquired from the NSSC Fitness Trail. 25,000 LIDAR data points were used per rectangle computation. The y axis points North, x axis points East.

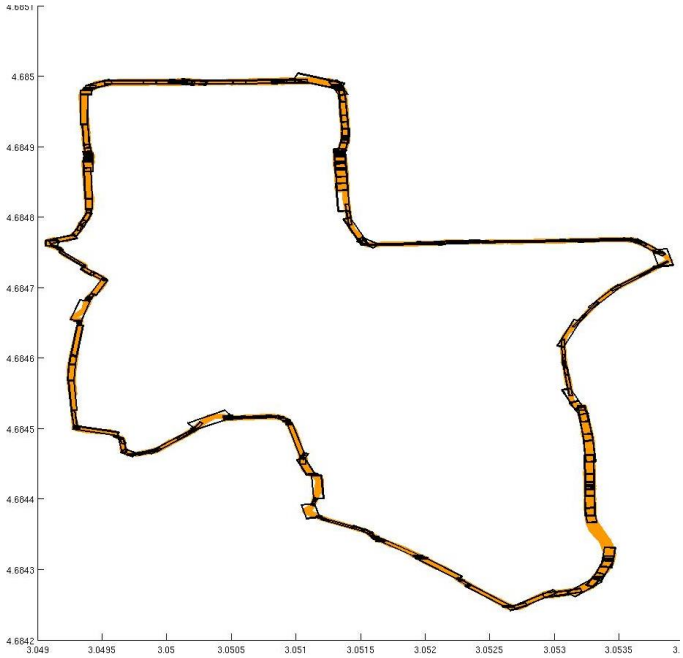

FIGURE 3. Minimum enclosing rectangles (black) from step 2 superimposed on the original LIDAR scanned ground point cloud data (orange). The y axis points North, x axis points East.

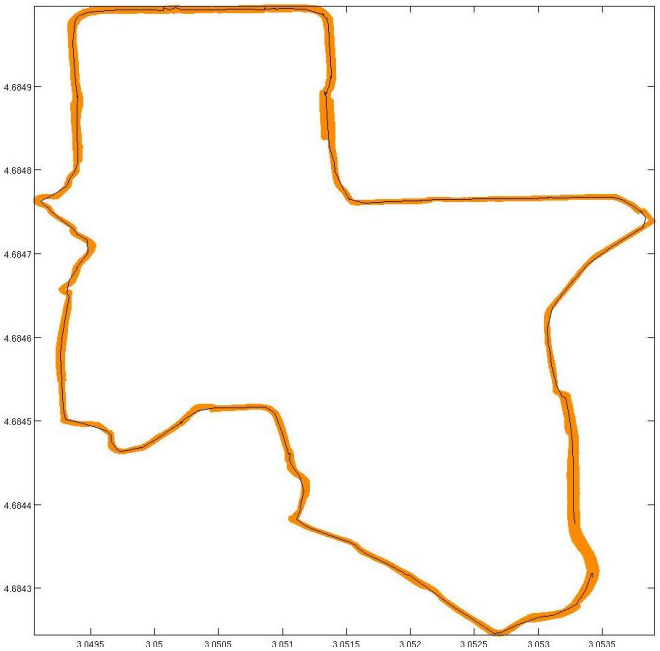

FIGURE 4. Unprocessed path through centroids from step 2 superimposed on the original LIDAR scanned ground point cloud data (orange). The y axis points North, x axis points East.

Prior to proceeding with the computer processing, the path formed by the centroids was manually edited to account for asymmetry in the acquired LIDAR ground scan data (e.g. due to right-left bias in the LIDAR data or vegetation). Any asymmetry in the collected data will cause the minimum rectangle-based method to create a path segment with centroids separated from the true path center, requiring manual editing.

It is then necessary to review the computed and edited rectangle centroids, and choose a minimum distance between the centroid locations that are retained. Centroid points at too close a distance are discarded. This would occur at locations of very densely positioned LIDAR data. This culling procedure is performed because of the variability in the density of scanned LIDAR locations (i.e. may range from low to very high density), and also the number of points chosen per set in step one (i.e. may be too few points chosen per rectangle yielding too many rectangles, hence too many centroids).

In cases where too many rectangles are computed, this step will have produced multiple rectangle centroids in close proximity due to overlapping centroids. These are reviewed in the code so that only path points separated by a specified minimum distance are retained.

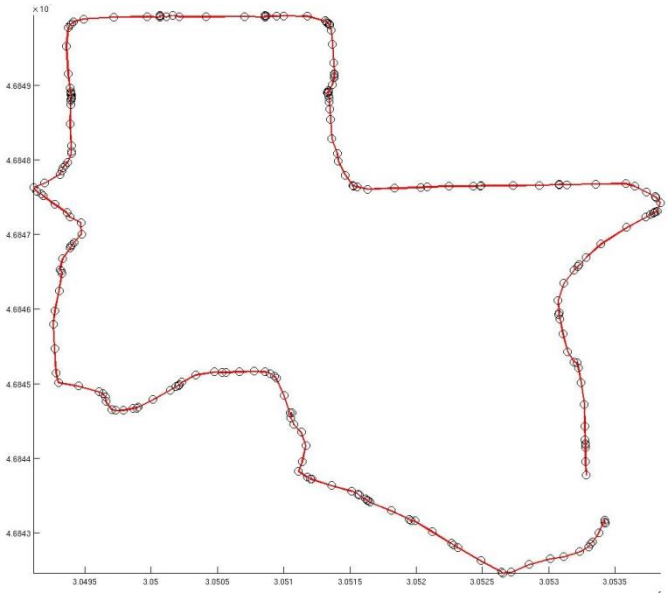

FIGURE 5. Centroids (black) after cull to retain minimum of 1.0m separation between centroid points, with resultant path (red) through centroids. The y axis points North, x axis points East.

DETERMINE PATH POINTS WITH UNIFORM SEPARATION

Because the output points of step two are unevenly separated, the final processing step is to determine the majority of path points at a uniform separation along the path formed by the rectangle centroids. From the path points determined previously, the first pair is considered, and the separation is computed. The number of increments of fixed length, as specified by the user, is computed in this curve segment. The x,y coordinates of the points forming the fixed length increments are computed trigonometrically assuming a linear path between the centroid coordinates, and the z coordinate is taken from the closest LIDAR scanned point measured in x,y.

Since only an integer number of fixed length increments may fit between successive centroid points, this approach may (in a large fraction of the instances) leave a segment remaining between centroids that is shorter than the user specified increment. The result is a path that extends through the centroids with separation between most intervening points equal to or less than a user chosen length when considering x,y. In actual computations, a large variation in z (the elevation), may cause a very small number of segments to exceed the specified segment separation by a small amount (Fig. 8).

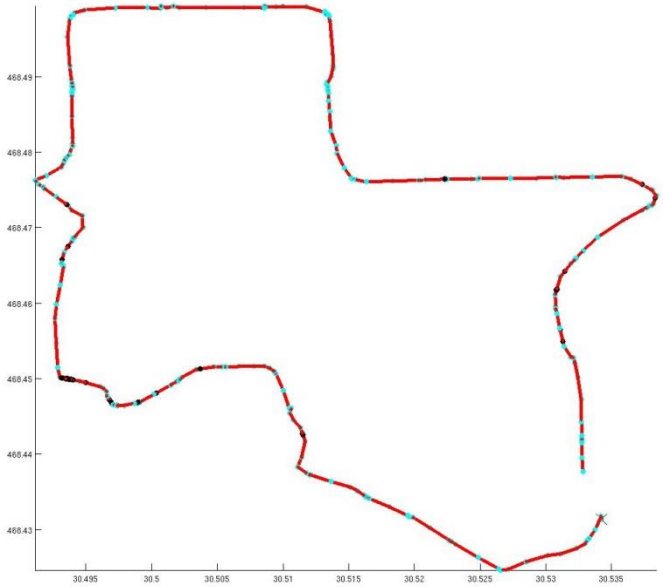

FIGURE 6. Final path with shorter (0.1m) fixed length increments (red) linking the centroids remaining from the previous step after the cull (cyan). The y axis points North, x axis points East.

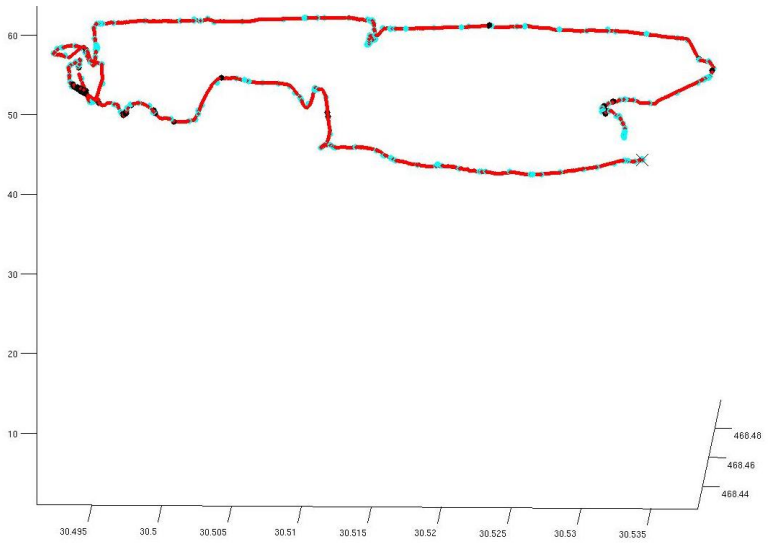

FIGURE 7. As FIG. 6, but from the Southwestern perspective showing elevations.

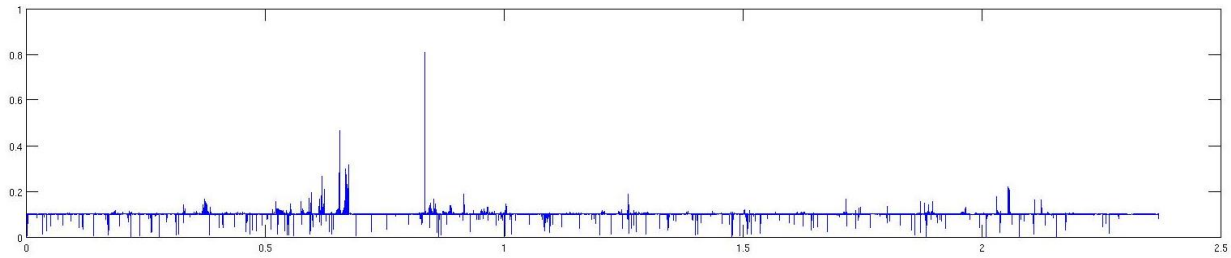

FIGURE 8. Distances (m), y axis, between points in the final path, x axis, showing the majority of points separated by 0.1m, smaller separations near rectangle centroids, and a small number of separations >0.1m due to significant changes in elevation that are ignored in specifying path point locations.

Next, if the data has been divided into sectors and subsequently reunited to form a single path, then there is a software check to eliminate the possibility of duplicate points being included from the boundaries of the sectors exhibiting zero separation. Then, if divided sectors have been reunited, the shorter fixed-length increments are added that link the start and end centroids.

Finally, a text file is created of the path x,y,z coordinates and r,g,b values.

RESULTS

The method was applied to determine the central path through LIDAR ground scan point cloud data acquired from the NSSC Fitness Trail having length approximately 1.5mi. This trail has been used in past research on the prediction of metabolic cost (Fig. 9). USGS LIDAR ground scan data consisted of 68,636,671 sets of x,y,z coordinates and r,g,b data acquired at variable (widely varying) density along the trail. All computations were performed on a Dell Precision T7500n computer with 12 Xeon CPUs and 47Gb DRAM.

FIGURE 9. NSSC Fitness Trail with location of specific fitness stations used in load carrying and metabolic cost research at USARIEM.

APPROXIMATE ORDERING ALONG PATH

In order to better tailor the input parameters to specific portions of the trail where point cloud density varied widely, the trail was divided into four sectors. Starting with the Northeast sector, trail coordinates were grouped within a circle of radius 2.5m from the starting trail coordinate chosen at the southeast most point in that sector. Data in that circle were stored in a coordinate list constituting output, and labeled signifying that they were not to be considered further in the approximate ordering process. The center of a new circle was then chosen at minimum distance (at approx. the distance between USGS coordinates) outside the previous circle, and the process is repeated using the eligible remaining coordinates.

The final output coordinate list is grouped in sets of coordinates ordered so as to lead from the starting trail coordinate to its end. The effect is to navigate the trail in the sector by large sets of points. This prepares this partly ordered coordinate list for the next step, which will search the trail for center coordinates. This same algorithm was used for all other sectors. Summarizing the circle radii used: 2.5m Northeast sector; 7.5m Southeast sector; 5.0m Southwest sector; and, 7.5m Northwest sector. Computation time was approximately 30 mins.

DETERMINING THE CENTER OF THE PATH

Processing the approximately ordered Northeast sector data, the minimum enclosing rectangle method entails initially choosing sets of coordinates. Summarizing the number of points considered to find the minimum enclosing rectangle, there were used: 2000 Northeast sector; 7500 Southeast sector; 5000 Southwest sector; and, 7500

Northwest sector. Due to the variation in the density of the points from sector to sector and within each sector, there can result overlapping rectangles determined by this algorithm step, and these will be dealt with in the following step. Processing the entire NSSC Fitness Trail dataset of 68.5 million data points in groups of 10,000 point sets, for example, yields approx. 6800 rectangles and centroids. Computation time was approximately 10 mins.

A minimum path point separation of 1m was imposed in this computational step, and used for all sectors. The path that results, notably, has points separated by various distances due to the non-uniform density of LIDAR ground scan points acquired.

The centroid data was then manually edited to account for deviations and artifacts. The centroids were overlaid on a 10 cm x 10 cm x 1 cm resolution version of the original LIDAR data with color. Next, three functional computer graphics tools were used to Insert, Replace and Remove points. Inserting and Replacing points would utilize points from the lower resolution rgb image. Then, both the lower resolution xyz and rgb LIDAR data and the centroid data was divided into the 40 separate timing gate-based sections that were used to clean the data previously. The timing gate sections refer to subsections of the trail used during past research to obtain intermediate passage times of participants.

Next, visually surveying individual sections, the calculated positions of the centroids were inspected, following the path that is traveled by experiment volunteer subjects. Very rarely were sections of LIDAR data appropriately proportioned on either side of the path, so the majority of points required minor manual adjustment. Additional points were added in areas of high curvature, such as corners and along curves, in order to properly capture the path shape that was smoothed by the automatic computational approach.

There was also a small section of the trail that was changed following the LIDAR scan due to new construction on site. In order to approximate that section of the trail, a screenshot image of the affected area was downloaded from Google Maps, and the centroids were overlaid manually. Using the Google maps image required a transformation of the data set that included a translation, scaling and 1 degree rotation, before returning it to its original UTM format. Following these adjustments the data was ready to be interpolated.

DETERMINE PATH POINTS WITH UNIFORM SEPARATION

The uniform path point separation of 10cm was used for all sectors. The result is a list of point coordinates in xyz sequentially moving clockwise around the trail, at 10cm separation. The rgb values from the nearest original USGS survey point data were found for each center path point, to create a text array totally 23,415 pts containing the xyz,rgb coordinates and values data sequentially listed to circumnavigate the trail. Path points found were: 4463 points Northeast sector; 8316 points Northwest sector; 4926 points Southwest sector; and, 5710 points Southeast sector. The 23,415 path points represent 2341.5m or 1.459mi. Computation time was approximately 4hrs.

CONCLUSION

This report describes a computational methodology for determining the central path through ground surface LIDAR ‘point cloud’ data, i.e. many points located in 3-D space. This approach answers the need for a singular point-to-point path useful for follow-up applications after acquisition of a large magnitude of data by LIDAR ground surveying. The processing of the data aims at a coarse-grained to fine-grained approach to locating the center of a local section of the overall path. This is meant to account for the

often prodigious amount of non-uniformly dense data produced by LIDAR ground scanning. A unique computational tool from the field of Computational Geometry is used to solve for the path coordinates in the presence of a large amount of variable density data and intervening obstructions. The centroids of the minimum enclosing rectangles that cover the LIDAR data are identified as final path points, since they are at the center of local LIDAR data and unaffected by the variable LIDAR data density. To provide a more precise description for later applications, the final path is defined to also include points uniformly interpolated between the rectangle centroid coordinates.

The approach that was developed was applied to USGS-scanned LIDAR coordinates and color data acquired from the Natick Soldier Systems Center (NSSC) Fitness Trail. Minor editing of the computed rectangle centroids was performed in order to account for deviations and artifacts in the acquired LIDAR data. The path determined from the results of this study will contribute directly to more accurate interpretations of warfighter physiology and performance experiments carried out using the trail

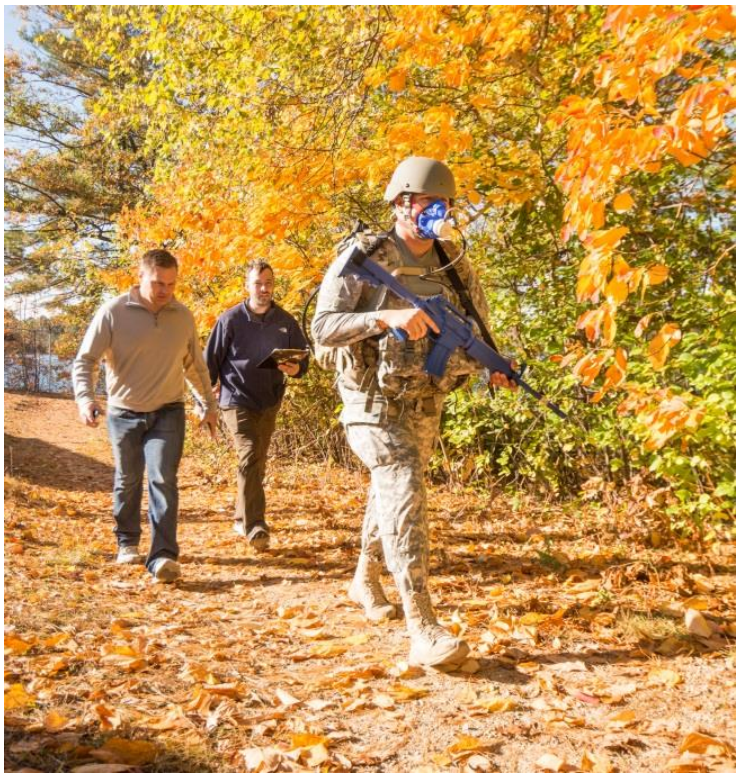

FIGURE 10. Research at USARIEM using the NSSC Fitness Trail. Researchers from the USARIEM Biophysics and Biomedical Modeling Division follow a load-carrying soldier test subject walking a circuit of the trail.

REFERENCES

Bellman R. On a routing problem. Quarterly of Applied Mathematics. 16: 87–90, 1958.

Cormen TH, Leiserson CE, Rivest RL, Stein C, *Introduction to Algorithms,* Second Edition. MIT Press and McGraw-Hill, 2001.

D'Errico J, A suite of minimal bounding objects, MathWorks File Exchange #34767, 2012.

Dijkstra EW. A note on two problems in connexion with graphs. Numerische Mathematik., 1:269–271, 1959.

Ford, L. R. (1956). *Network Flow Theory.* Rand Corporation. P-923.

Preparata FP, Shamos M, *Computational Geometry: An Introduction*, Monographs in Computer Science, Springer, 1985.

Toussaint GT, Solving Geometric Problems with the "Rotating Calipers," Proc of IEEE MELECON 83, Athens, Greece, 1983.

MATLAB COMMANDS

INTRODUCTION

```
fid = fopen('Raw_path_data', 'r');
Data_orig = textscan(fid, '%f %f %f %d %d %d');  % Data_orig is a cell array
fclose(fid);
Data_xyz = [Data_orig{1,1}; Data_orig{1,2}; Data_orig{1,3}];
Data_rgb = [Data_orig{1,4}; Data_orig{1,5}; Data_orig{1,6}];
Data_xyz = Data_xyz(randperm(size(Data_xyz,1)),:);
```

APPROXIMATE ORDERING ALONG PATH

```
Data_step1 = get_rearranged_path(Data_xyz, 7.5, 2, 2);  % signifies start at bottom
right sector of the data, right side, using circles radius 7.5m
```

DETERMINING THE CENTER OF THE PATH

```
[Rx, Ry, Rz, Area, Perimeter] = get_path_walk(Data_step1, 5000); % here using 5000
coordinates per set considered

[Route] = get_path_simple_route(Rx, Ry, Rz, 1.0); % from the minimum rectangle
vertices, compute the centroid to be a point on our path
```

DETERMINE PATH POINTS WITH UNIFORM SEPARATION

```
RefRoute = get_refined_route3B_with_color(Route, 0.1, Axyz, Argb);

RefRoute = get_rid_of_zeros(RefRoute);

RefRoute = get_last_segment(RefRoute, 0.10, Axyz, Argb);

get_final_textfile(RefRoute, 'Final_route.txt');
```

MATLAB CODE

```
function [Aout] = get_rearranged_path(A, search_radius, nstart_sector,
nstart_side)
% consider pts nearby within search_radius meters (approx 5)
% nstart sector = 1,2,3,4 numbered cw from top right(1), bottom
right(2), bottom left(3), top left(4)
% nstart side = 1,2,3,4 numbered cw from top(1), right(2), bottom(3),
left(4)

Tmean = mean(A); xmean = Tmean(1); ymean = Tmean(2); zmean = Tmean(3);
sprintf('xmean x10-4: %f  ymean x10-4: %f  zmean: %f', xmean*0.0001,
ymean*0.0001, zmean)

na = size(A,1);
Active = true(na,1);
Aout = zeros(na,3);

figno = 17;
 if ishghandle(figno)
  close(figno);
end;

search_radius2 = search_radius * search_radius;
% find all pts with search_radius meters of focal point

% choose starting location
switch nstart_sector
  case 1
   Ks = find((A(:,1) > xmean) & (A(:,2) > ymean));
  case 2
   Ks = find((A(:,1) > xmean) & (A(:,2) < ymean));
  case 3
   Ks = find((A(:,1) < xmean) & (A(:,2) < ymean));
  case 4
   Ks = find((A(:,1) < xmean) & (A(:,2) > ymean));
 otherwise
  sprintf('Bad nstart_sector value %d enetered. Must be 1,2,3,4',
start_sector)
end

switch nstart_side
  case 1
   [Y, Istart] = max(A(Ks,2)); Istart = Ks(Istart);
  case 2
   [Y, Istart] = max(A(Ks,1)); Istart = Ks(Istart);
  case 3
   [Y, Istart] = min(A(Ks,2)); Istart = Ks(Istart);
  case 4
   [Y, Istart] = min(A(Ks,1)); Istart = Ks(Istart);
  otherwise
```

```
  sprintf('Bad nstart_side value %d enetered. Must be 1,2,3,4',
start_side)
end

ifocus = Istart;  Active(ifocus)=false;
 iout = 1; icount= 1;
% look only at deviations to account for large and small GPS values
A(:,1) = A(:,1) - xmean;   A(:,2) = A(:,2) - ymean;  A(:,3) = A(:,3) -
zmean;

figure(figno); plot(A(ifocus,1), A(ifocus,2), 'rx', 'MarkerSize', 20);
axis tight;  %  axis( [min(A(:,1)) max(A(:,1)) min(A(:,2))
max(A(:,2))]);
% ------------------------------------------------------------------
% ------------------------------------------------------------------

while (ifocus<na) & (sum(Active)>0)
  Iact = find(Active); % remaining vertices in A not yet preordered
  clear D
  D = (A(ifocus,1) - A(Iact,1)).* (A(ifocus,1) - A(Iact,1)) +
(A(ifocus,2) - A(Iact,2)).* (A(ifocus,2) - A(Iact,2)) +   (A(ifocus,3)
- A(Iact,3)).* (A(ifocus,3) - A(Iact,3));
%      D=sqrt(D);

  K = find(D<=search_radius2); K0 = K;  % indicies of Iact
  Active(ifocus) = false;

   figure(figno); hold on; plot(A(ifocus,1), A(ifocus,2), 'bo',
'MarkerSize', 20);  axis tight; hold off; drawnow;  %  axis(
[min(A(:,1)) max(A(:,1)) min(A(:,2)) max(A(:,2))]);
 fprintf(1,'Active: %d ifocus: %d size(D): %d %d size(Iact): %d %d
size(K): %d %d \n', sum(Active), ifocus, size(D), size(Iact),
size(K));
 % ----------------------------------

 if(~isempty(K))
  K = Iact(K);    % K store indicies into A of nearby vertices (within
search_radius)

  % fill in Aout
  Aout(iout:(iout+(size(K,1)-1)),:) = [(A(K,1) + xmean)  (A(K,2) +
ymean)  (A(K,3) + zmean)];
  iout = iout + size(K,1);  % index in Aout for filling start
  Active(K) = false;   D(K0) = 1.0e+25;
 end
 % ----------------------------------

  % find the index of the min D to be next focus
   [Y, I] = min(D);
  fprintf(1,'                          size(D):%d %d  size(Iact): %d %d
Y: %f  I: %d\n', size(D), size(Iact), Y, I)
  ifocus = Iact(I);  Active(ifocus)=false;
```

```
  clear Iact D K I
  icount = icount + 1;
end;

% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
return;
```

```
function [Rx, Ry, Rz, Area, Perimeter] = get_path_walk(A,
npts_per_step)

perimeter_limit = 100;
nsteps = floor(size(A,1)/npts_per_step);
Area = zeros(nsteps*2,1); Perimeter = zeros(nsteps*2,1);
Rx = zeros(nsteps*2,4);   Ry = zeros(nsteps*2,4);  Rz =
zeros(nsteps*2,4);

figno = 15;
 if ishghandle(figno)
  close(figno);
end;

figure(figno); plot(A(1,1), A(1,2), 'ro');
%  view(0,90);
% ------------------------------------------------------------------

nstart = 1;
nend = npts_per_step;
k=1;

while nend<=size(A,1)
  [rectx, recty, xarea, xperimeter] = minboundrect(A(nstart:nend,1),
A(nstart:nend,2));
  % store in rectz the z value of the actual vertex closest to the
centroid of the rectangle
  xcen = mean(rectx); ycen = mean(recty);
  Izcen = get_closest_trailpoint(A(nstart:nend,:), [xcen ycen]); Izcen
= nstart + Izcen - 1; rectz = A(Izcen,3);

  if (xperimeter>perimeter_limit)
    while (xperimeter>perimeter_limit) & (nend > (nstart + 10))

     ninc = floor((nend - nstart + 1)/2);
     nend = nstart + ninc;
     [rectx, recty, xarea, xperimeter] =
minboundrect(A(nstart:nend,1), A(nstart:nend,2));
     xcen = mean(rectx); ycen = mean(recty);
     Izcen = get_closest_trailpoint(A(nstart:nend,:), [xcen ycen]);
Izcen = nstart + Izcen - 1; rectz = A(Izcen,3);
    end;
  end

% ----- here if xperimeter<perimeter_limit -----

  Area(k) = xarea; Perimeter(k) = xperimeter;
  Rx(k,:) = rectx(1:4)'; Ry(k,:) = recty(1:4)'; Rz(k,:) = [rectz rectz
rectz rectz];
  figure(figno); hold on; plot(rectx, recty, 'r-');
plot(mean(rectx(1:4)), mean(recty(1:4)), 'g.', 'MarkerSize', 10); hold
off; drawnow;
```

```
 k=k+1;
 nstart = nend+1;
 nend = nstart + npts_per_step - 1;
end;   %  end of while
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------

ktot = k-1;

% ----- draw with markers -----
dothis = false;
if dothis
for k=1:ktot;
  rectx = Rx(k,:);  recty = Ry(k,:);
  figure(figno); hold on; plot(mean(rectx), mean(recty), 'c.',
'MarkerSize', 10); hold off; drawnow;
end;
end

Rx = Rx(1:ktot,:); Ry = Ry(1:ktot,:); Rz = Rz(1:ktot,:); Area =
Area(1:ktot); Perimeter = Perimeter(1:ktot);
figure(figno); axis tight;
figno = 16;
 if ishghandle(figno)
  close(figno);
end;
figure(figno); plot(nonzeros(A(:,1)), nonzeros(A(:,2)), 'r.',
'MarkerSize', 5, 'Color', [1.0 140/255 0]);
figure(figno); hold on; plot(mean(Rx')', mean(Ry')', 'k-');  hold off;
axis tight;  drawnow;
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
return;
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------

function [I] = get_closest_trailpoint(A, point_in_space)
% get the vertex in A that is closest to [x y] only of point_in_space
where point in space is [x y]
%   ie compute as if coplanar pts

d1  = A(:,1) - point_in_space(1);
d2  = A(:,2) - point_in_space(2);
dist = d1.*d1 + d2.*d2; clear d1 d2 d3
[mindist, I] = min(dist);
return;
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
```

```
function [Route] = get_path_simple_route(Rx, Ry,  Rz, minlength)
% choose the path where pts are a minimum of minlength distant from
each other

Oroute = [mean(Rx')' mean(Ry')' mean(Rz')'];   nor = size(Oroute,1);
Active = true(nor,1);

figno = 18;
 if ishghandle(figno)
  close(figno);
end;

D = zeros(nor, nor);  Route = zeros(nor,3);  icount = 1;
k = 1;

while (k <= nor)
 if Active(k)
  D = ((Oroute(k,1) - Oroute(:,1)).* (Oroute(k,1) - Oroute(:,1))  +
(Oroute(k,2) - Oroute(:,2)).* (Oroute(k,2) - Oroute(:,2)) +
(Oroute(k,3) - Oroute(:,3)).* (Oroute(k,3) - Oroute(:,3)));
  D = sqrt(D);
  J = find(~Active);  D(J) = inf;
  K = find(D<minlength);

  if size(K,1)>1
   % in here if more than one pt in cluster
   Route(icount,:) = [mean(Oroute(K,1))    mean(Oroute(K,2))
mean(Oroute(K,3)) ];
   Active(K)=false;
  else
   Route(icount,:) = Oroute(k,:);  Active(k)= false;
  end

  icount = icount + 1;
 end
 k = k + 1;
end;

Route = Route(1:(icount-1),:);
figure(figno); hold on; plot3(Route(:,1), Route(:,2), Route(:,3),
'ko', 'MarkerSize', 10); drawnow;
view(0,90); axis tight;

% draw a curve thru the red circles
figure(figno); hold on; plot3(Route(:,1), Route(:,2), Route(:,3),  'r-
', 'LineWidth', 2); axis([min(Route(:,1)) max(Route(:,1))
min(Route(:,2)) max(Route(:,2)) 1 75]);  drawnow;
return;
% ------------------------------------------------------------------
% ------------------------------------------------------------------
```

```
function [RefRoute] = get_refined_route3B_with_color(Route_xyz,
deltaL, Axyz, Argb)
% METHOD #3  THREE B - fit delta increments between centroids & always
%  start next segment with an increment starting at a centroid

%  get a refined route thru the path shown in Route with pts separated
by deltaL meters - assumes all measures in meters
% 1yd = 0.9144m
% 1 mi = 1609.3m or 160930cm

%  input:
% Route_xyz - x,y,z pts from computation and manual editing
% deltaL - distance of final pts along path  IN METERS
% Axyz, Argb - orig laser scanned x,y,z coords and r,g,b color - must
be Axyz here since ordered similar to Argb

Home = zeros(1,3);
if ishghandle(20)
  close(20);
end
if ishghandle(22)
 close(22);
end

kref = 1;
figure(20); plot3( 1.0e-04*Route_xyz(:,1),   1.0e-04*Route_xyz(:,2),
Route_xyz(:,3), 'c.', 'Markersize', 20); view(0,90);
hold on; plot3( 1.0e-04*Route_xyz(1,1),   1.0e-04*Route_xyz(1,2),
Route_xyz(1,3), 'kx', 'Markersize', 20); view(0,90);
axis([min(0.0001*Route_xyz(:,1))  max(0.0001*Route_xyz(:,1))
min(0.0001*Route_xyz(:,2))  max(0.0001*Route_xyz(:,2))  1 65]);
hold off; drawnow;

RefRoute = zeros(10000,6);
% consider all pairs of points
% fix origin at 'home' point - 'home' point starts at Route(1,:)
Home(1:3) = Route_xyz(1,1:3);
% ------------------------------------------------------------------
% ------------------------------------------------------------------

for k=2:size(Route_xyz,1);

  % 3D distance from home pt to next route pt #k
  dist_hr = sqrt( (Home(1)-Route_xyz(k,1)).*(Home(1)-Route_xyz(k,1)) +
(Home(2)-Route_xyz(k,2)).*(Home(2)-Route_xyz(k,2)) +  (Home(3)-
Route_xyz(k,3)).*(Home(3)-Route_xyz(k,3)) );

  incs = floor(dist_hr / deltaL);  % MTD 1 & MTD 3   - increments of
size deltaL wholy fitting from home to route pt
%   incs = ceil(dist_hr / deltaL);  % MTD 2 - increments of size
deltaL wholy fitting from home to route pt
 if incs>0
```

```
  Q = zeros(500,3);
  deltaX = Route_xyz(k,1) - Home(1);   deltaY = Route_xyz(k,2) -
Home(2);  deltaZ = Route_xyz(k,3) - Home(3);  deltaS = deltaL/dist_hr;
% sprintf('del x,y,z: %f %f %f deltaS,incs: %f %d incs*deltaS: %f',
deltaX, deltaY, deltaZ, deltaS, incs, (incs*deltaS) )

  for m=0:incs;
%   Q(1:(incs+1),:) = [(Home(1)+deltaX*deltaS.*(0:incs)')   (Home(2) +
deltaY*deltaS.*(0:incs)')    (Home(3) + deltaZ*deltaS.*(0:incs)') ];
% new path pts deltaL separation

    Q((1+m),:) = [(Home(1)+m*deltaX*deltaS)   (Home(2) +
m*deltaY*deltaS)    (Home(3) + m*deltaZ*deltaS) ];
   RefRoute((kref+m),1:3) =  [(Home(1)+m*deltaX*deltaS)   (Home(2) +
m*deltaY*deltaS)    (Home(3) + m*deltaZ*deltaS) ];
  end;
   % ------------------------------------------------------

% update
  kref = kref + incs + 1;

 else
   % found incs==0 - so skip this Route pt and move to next, keeping
home the same
   sprintf('k:%d found incs=0 Home: %f %f %f', k, Home(1:3))
 end  % to if incs > 0
 Home = Route_xyz(k,:); % MTD 3 actual gold std pt is used as home
 end; % of for k
   % ------------------------------------------------------

 kref = kref - 1;
 RefRoute = RefRoute(1:kref,:);
% ------------------------------------------------------
%  add z information
dothis=true;
if dothis
  for k=1:kref;

   if mod(k,100)==0
     sprintf('k: %d   kref: %d', k, kref)
   end

  % get the z & rgb from the closest original data point
    I = get_closest_original_point3(RefRoute(k,1:3), Axyz);

    RefRoute(k,3) = Axyz(I,3);
    RefRoute(k,4:6) =  double(Argb(I,:));
  figure(20);  hold on; plot3( 1.0e-04*RefRoute(k,1), 1.0e-
04*RefRoute(k,2), RefRoute(k,3), 'r.', 'Markersize', 5); view(0,90);
hold off; drawnow;
  end;
```

```
end
% --------------------------------------------------------------------
% --------------------------------------------------------------------

% check on distance between pts
%  dist_pts = sqrt ( diff(RefRoute(:,1)).*diff(RefRoute(:,1)) +
diff(RefRoute(:,2)).*diff(RefRoute(:,2)) +
diff(RefRoute(:,3)).*diff(RefRoute(:,3)) );

n = size(RefRoute,1);
dist_pts = sqrt( (RefRoute((2:n),1)-RefRoute((1:(n-
1)),1)).*(RefRoute((2:n),1)-RefRoute((1:(n-1)),1)) +
(RefRoute((2:n),2)-RefRoute((1:(n-1)),2)).*(RefRoute((2:n),2)-
RefRoute((1:(n-1)),2)) +  (RefRoute((2:n),3)-RefRoute((1:(n-
1)),3)).*(RefRoute((2:n),3)-RefRoute((1:(n-1)),3)) );

total_distance = sum(dist_pts);   % in meters
% add the last gap
total_distance = total_distance +  sqrt( (RefRoute(n,1)-
RefRoute(1,1)).*(RefRoute(n,1)-RefRoute(1,1)) +  (RefRoute(n,2)-
RefRoute(1,2)).*(RefRoute(n,2)-RefRoute(1,2)) +  (RefRoute(n,3)-
RefRoute(1,3)).*(RefRoute(n,3)-RefRoute(1,3)) );

sprintf('Total length of path: %f meters %f yards %f miles',
total_distance, (total_distance/0.9144),
((total_distance/0.9144)/1760) )
figure(22); plot(dist_pts);

K = find(dist_pts>0.15);
figure(20);  hold on; plot3( 1.0e-04*RefRoute(K,1), 1.0e-
04*RefRoute(K,2), RefRoute(K,3), 'k.', 'Markersize', 20);
hold off; drawnow;
% --------------------------------------------------------------------
% --------------------------------------------------------------------
return;
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --------------------------------------------------------------------
```

```
function [RefRoute_out] = get_rid_of_zeros(RefRoute)
% remove duplicate points
% check on distance between pts
n = size(RefRoute,1);  RefRoute_out = zeros(size(RefRoute,1),6);
new=1;

dist_pts = zeros(size(RefRoute,1),1);
for k=1:(n-1);
 dist_pts(k) = sqrt( (RefRoute(k,1)-RefRoute((k+1),1))*(RefRoute(k,1)-
RefRoute((k+1),1)) +  (RefRoute(k,2)-
RefRoute((k+1),2))*(RefRoute(k,2)-RefRoute((k+1),2)) +
(RefRoute(k,3)-RefRoute((k+1),3))*(RefRoute(k,3)-RefRoute((k+1),3)) );
  if dist_pts(k)==0
   sprintf('Encountered 0 dist between k: %d & %d', k, (k+1))
  else
   RefRoute_out(new,:) = RefRoute(k,:); new=new+1;
 end

end;

dist_pts(n) = sqrt( (RefRoute(1,1)-RefRoute(n,1))*(RefRoute(1,1)-
RefRoute(n,1)) +  (RefRoute(1,2)-RefRoute(n,2))*(RefRoute(1,2)-
RefRoute(n,2)) +  (RefRoute(1,3)-RefRoute(n,3))*(RefRoute(1,3)-
RefRoute(n,3)) );

  if dist_pts(n)==0
   sprintf('Encountered 0 dist between k: 1 & n')
  else
   RefRoute_out(new,:) = RefRoute(n,:); new=new+1;
 end

new = new - 1;
RefRoute_out = RefRoute_out(1:new,:);
total_distance = sum(dist_pts);   % in meters

sprintf('Total length of path: %f meters %f yards %f miles',
total_distance, (total_distance/0.9144),
((total_distance/0.9144)/1760) )
% ------------------------------------------------------------------
% ------------------------------------------------------------------
return;
% ------------------------------------------------------------------
% ------------------------------------------------------------------
% ------------------------------------------------------------------
```

```
function [RefRoute] = get_last_segment(Route_in, deltaL, Axyz, Argb)
% based on METHOD #1  ONE  B
%  get a refined route thru the path shown in Route with pts separated
by deltaL meters - assumes all measures in meters
% 1yd = 0.9144m
% 1 mi = 1609.3m or 160930cm

%  input:
% Route_xyz - x,y,z pts from computation and manual editing
% deltaL - distance of final pts along path  IN METERS
% Axyz, Argb - orig laser scanned x,y,z coords and r,g,b color

Home = zeros(1,3);
if ishghandle(20)
  close(20);
end
if ishghandle(22)
 close(22);
end

kref = 1;
RefRoute = [Route_in; zeros(50,6)];
n = size(Route_in,1);
% fix origin at 'home' point
Home(1:3) = Route_in(n,1:3);
% ------------------------------------------------------------------
% ------------------------------------------------------------------

  % 3D distance from home pt to next route pt #1
  dist_hr = sqrt( (Home(1)-Route_in(1,1)).*(Home(1)-Route_in(1,1)) +
(Home(2)-Route_in(1,2)).*(Home(2)-Route_in(1,2)) +  (Home(3)-
Route_in(1,3)).*(Home(3)-Route_in(1,3)) );

  incs = floor(dist_hr / deltaL);  % MTD 1 & MTD 3   - increments of
size deltaL wholy fitting from home to route pt
%   incs = ceil(dist_hr / deltaL);  % MTD 2 - increments of size
deltaL wholy fitting from home to route pt

  if ((dist_hr - incs*deltaL) < 0.5*deltaL)
   incs = incs - 1;
  end

  if incs>0
  deltaX = Route_in(1,1) - Home(1);   deltaY = Route_in(1,2) -
Home(2);  deltaZ = Route_in(1,3) - Home(3);  deltaS = deltaL/dist_hr;

  for m=1:incs;
   RefRoute((n+m),1:3) =  [(Home(1)+m*deltaX*deltaS)   (Home(2) +
m*deltaY*deltaS)    (Home(3) + m*deltaZ*deltaS) ];
  end;

   % ------------------------------------------------------
```

```
 else
   % found incs==0 - so skip this Route pt and move to next, keeping
home the same
   sprintf('k:%d found incs=0 Home: %f %f %f', k, Home(1:3))
 end  % to if incs > 0

 RefRoute = RefRoute(1:(n+incs),:);
% ------------------------------------------------------

%  add z information
dothis=true;
if dothis
  for k=(n+1):(n+incs);
  % get the z & rgb from the closest original data point
    I = get_closest_original_point3(RefRoute(k,1:3), Axyz);

    RefRoute(k,3) = Axyz(I,3);
    RefRoute(k,4:6) =  double(Argb(I,:));
  end;
end
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% ------------------------------------------------------------

% check on distance between pts
n = size(RefRoute,1);

dist_pts = zeros(size(RefRoute,1),1);
for k=1:(n-1);
 dist_pts(k) = sqrt( (RefRoute(k,1)-RefRoute((k+1),1))*(RefRoute(k,1)-
RefRoute((k+1),1)) +  (RefRoute(k,2)-
RefRoute((k+1),2))*(RefRoute(k,2)-RefRoute((k+1),2)) +
(RefRoute(k,3)-RefRoute((k+1),3))*(RefRoute(k,3)-RefRoute((k+1),3)) );

  if dist_pts(k)==0
   sprintf('Encountered 0 dist between k: %d & %d', k, (k+1))
  end
end;

dist_pts(n) = sqrt( (RefRoute(1,1)-RefRoute(n,1))*(RefRoute(1,1)-
RefRoute(n,1)) +  (RefRoute(1,2)-RefRoute(n,2))*(RefRoute(1,2)-
RefRoute(n,2)) +  (RefRoute(1,3)-RefRoute(n,3))*(RefRoute(1,3)-
RefRoute(n,3)) );

  if dist_pts(n)==0
   sprintf('Encountered 0 dist between k: 1 & n')
 end

total_distance = sum(dist_pts);   % in meters
```

```
sprintf('Total length of path: %f meters %f yards %f miles',
total_distance, (total_distance/0.9144),
((total_distance/0.9144)/1760) )
figure(22); plot(dist_pts);
% --------------------------------------------------------------------
% --------------------------------------------------------------------
return;
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --------------------------------------------------------------------
```

```
function [] = get_final_textfile(Route, filename)
% create the final text file on disk
fid = fopen(filename, 'w');

for k=1:size(Route,1);
  fprintf(fid,'%11.3f %11.3f %6.3f %3d %3d %3d\n', Route(k,1:3),
Route(k,4:6));
end;

fclose(fid);
return;
% --------------------------------------------------------------------
% --------------------------------------------------------------------
% --------------------------------------------------------------------
```